Sun Identity Manager 8.1 Web Services

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–5597
February 2009

090312@21990

Contents

Preface

Sun Identity Manager 8.1 Web Services contains reference and procedural information designed to help you use SPML Versions 1.0 or 2.0 to communicate with Sun™Identity Manager and SunIdentity Manager Service Provider service provisioning activities. Specifically, this document describes which features are supported and why, how to configure SPML support, and how to extend support in the field.

Who Should Use This Book

This book is for application developers and developers who are responsible for deploying Identity Manager, implementing procedural logic, and using SPML classes to format service provisioning request messages and to parse response messages.

Related Web Site

For information about using OpenSPML, see `http://www.openspml.org`.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

◆ ◆ ◆ CHAPTER 1

Using SPML 1.0 With Sun Identity Manager Web Services

Service Provisioning Markup Language (SPML) 1.0 is an OASIS standard used to provide an open interface for communicating with service provisioning activities. You access Sun™ Identity Manager Web Services by using SPML requests for HTTP.

This chapter describes SPML 1.0 support in Sun Identity Manager software (Identity Manager) and Sun Identity Manager Service Provider (Service Provider) and includes information about which features are supported and why, how to configure SPML 1.0 support, and how to extend support in the field.

The information is organized as follows:

Note – Identity Manager supports both SPML Version 1.0 and Version 2.0. The concepts described in this chapter *relate specifically to SPML 1.0*. However, this information also provides a good basis for understanding concepts described in Chapter 2, "Using SPML 2.0 With Sun Identity Manager Web Services."

Important Notes About Using SPML 1.0

Before you start working with SPML 1.0 and Identity Manager Web Services, note the following:

- For optimal performance when you are working with the Identity Manager Web Services interfaces, use the OpenSPML Toolkit that is co-packaged with Identity Manager. Using the `openspml.jar` file from the `http://www.openspml.org` web site might cause memory leaks.
- The Service Provider REF Kit contains an `SpmlUsage.java` file that demonstrates how to use the Service Provider SPML interface.
- You can access Service Provider features through SPML 1.0. (These features are not available with SPML Version 2.0.)

  The Service Provider SPML interface is very similar to the Identity Manager SPML interface. Differences in configuration and operation are noted in this chapter where appropriate.

Configuring SPML

To expose the SPML interface, you must properly configure the Identity Manager server by installing and modifying specific repository objects and by editing the `Waveset.properties` file.

Instructions for configuring the SPML interface are provided in the following sections:

- "Installing and Modifying Repository Objects" on page 10
- "Editing the `Waveset.properties` File" on page 12
- "Editing Configuration Objects" on page 14

Installing and Modifying Repository Objects

The following table describes the repository objects that you must install and modify to configure SPML for Identity Manager.

TABLE 1–1 Repository Objects Used to Configure SPML

Object	Description
`Configuration:SPML`	Contains definitions of the SPML schemas supported by the server, and rules for converting between the SPML schema and the internal view model. Each SPML schema typically has an associated form.

TABLE 1–1 Repository Objects Used to Configure SPML *(Continued)*

Object	Description
SPML Forms	Contains one or more form objects that encapsulate the rules for transforming between the external model (defined by an SPML schema) and the internal model (defined by an Identity Manager view). Typically, you define one SPML form for each object class defined in the SPML schema.
`Configuration:IDM Schema Configuration`	Defines user attributes that can be stored in the Identity Manager repository for access through an SPML filter, and which are *queryable* and *summary* attributes for Identity Manager user objects. ■ Define a queryable attribute for attributes that you want to use in an SPML filter. ■ Define a summary attribute for attributes that you want returned in an optimized search.
`TaskDefinition:SPMLRequest`	System task used to process asynchronous SPML requests. You typically do not have to customize this object.

Identity Manager includes a sample set of SPML configuration objects in the `sample/spml.xml` file. You must manually import the `sample/spml.xml` file because the file is not imported by default when the repository is initialized.

The sample configuration defines a `person` class to track the evolving standard schema defined by the SPML working group. *Do not customize this class.* You must keep the `person` class consistent with the standard schema, *except* when you are configuring the Service Provider SPML interface.

When configuring the Service Provider SPML interface, you must install and modify the `Configuration:SPE SPML` configuration object as follows:

- Configure the `person` class (the only object class defined by default) to use the Service Provider-specific view handler (`IDMXUser`).
- Use the `form` attribute to define a user form that translates between the SPML request or SPML response and the view.

  The `form` attribute can take a special value `(view):` in which no form processing is applied to the view. For example, the `view` is passed directly between the client and Identity Manager.

You access the Service Provider SPML interface from the following (default) path:

```
/servlet/spespml
```

For example, if you deploy Identity Manager in the `/idm` context on *host*:*port*, you can access the interface at the following URL:

```
http://host:port/idm/servlet/spespml
```

where:

- *host* is the machine on which you are running Identity Manager.
- *port* is the number of the TCP port on which the server is listening.

Note – See the SPML 1.0 Specification at `http://www.openspml.org` for the most current information about the standard SPML schema.

Editing the `Waveset.properties` File

The following table describes three optional entries in the `Waveset.properties` file that you can use to control how SPML requests are authorized.

TABLE 1–2 Optional Entries in `Waveset.properties`

Entry Name	Description
`soap.username`	Name of the Identity Manager user who performs SPML requests
`soap.password`	Clear text password for the user specified by `soap.username`
`soap.epassword`	Base-64 representation of an encrypted password for the user specified by `soap.username`

Editing `soap.epassword` and `soap.password` Properties

The user specified in `soap.username` is known as the *proxy user*.

You can specify only one password property for the proxy user:

- Specifying `soap.password` is the simplest option, but this property exposes a clear text password in the `properties` file.
- Specifying `soap.epassword` is a more secure option, but you must perform extra steps to generate an encrypted password.

Establishing a proxy user is convenient for clients because authentication is not required by the web service. This configuration is common for portal environments where the Identity Manager server is only accessed by other applications that handle user authentication.

Caution – Using a proxy user can be dangerous if the HTTP port on which the responding server resides is generally accessible. Anyone who knows the Identity Manager server's URL and understands how to build SPML requests can configure Identity Manager operations for the proxy user to perform.

The SPML standard does not specify how to perform authentication and authorization. Several related web standards are available for authentication, but these standards are not yet in common use. At this time, the most common approach for authentication is to use the Secure Socket Layer (SSL) between applications and the server. Identity Manager does not dictate how to configure SSL.

If you cannot use a proxy user or SSL, Identity Manager supports a vendor-specific extension to SPML that allows the client to log in and maintain a session token, which can be used to authenticate subsequent requests. You can use the `LighthouseClient` class (an extension of the `SpmlClient` class that includes support for specifying credentials) to perform a login request and pass a session token in all SPML requests.

Note – The Service Provider SPML interface does not support authentication and authorization. However, you can configure the Identity Manager SPML interface to use the `IDMXUser` view instead of using Service Provider SPML.

Service Provider assumes that clients accessing Identity Manager have been authenticated and authorized by an access management application. The client has all possible rights when using the Service Provider SPML interface.

To prevent sensitive data from being exposed between the client and Identity Manager, consider accessing the Service Provider SPML interface over SSL.

Creating an Encrypted Password

Use one of the following methods to create an encrypted password:

- Open the Identity Manager console and use the `encrypt` command.
- Open the Identity Manager Debug pages or console and view the XML for the proxy user. Find the `WSUser` element for the `password` attribute value and use that value for the `soap.epassword` property.

  Note – To access the Debug pages, open the Identity Manager Administrator interface and type the following URL:

  `http://`*host*`:`*port*`/idm/debug`

  where *host* is the local server where Identity Manager is running, and *port* is the TCP port on which the server is listening.

Editing Configuration Objects

Applications require a mechanism to send SPML messages and to receive SPML responses.

To configure SPML for Identity Manager, you must edit the following configuration objects:

Note – The Service Provider SPML interface has only one configuration object, `Configuration:SPE SPML`, which is similar to the `Configuration:SPML` object in structure.

`Configuration: SPML` Object

The `Configuration: SPML` object contains definitions for the SPML schemas that you want to expose, and information about how those SPML schemas are mapped into Identity Manager views. This information is represented by using a `GenericObject` that is stored as an extension of the configuration object.

The following attributes are defined in `GenericObject`: `schemas` and `classes`:

- **Schemas**. A list of strings, where each string contains the escaped XML for one SPML `<schema>` element. Because the SPML elements are not defined in the `waveset.dtd`, you cannot directly include them in an Identity Manager XML document. Instead, you must include them as escaped text.
- **Classes**. A list of objects containing information about the supported SPML classes and how those classes are mapped into views. Define one object from this list for each class defined by the SPML schemas on the `schemas` list.

Initially, the distinction between the two lists might be confusing. The information in the *schemas* list defines what Identity Manager returns in response to an SPML `SchemaRequest` message. The client uses this information to decide which attributes can be included in other messages such as `AddRequest`. Identity Manager does not care about the contents of the schemas list. This list is simply returned verbatim to the client.

You are not required to define SPML schemas. Identity Manager works without schemas. If you do not define an SPML schema, Identity Manager returns an empty response after receiving a schema request message. Without a schema, clients must rely on pre-existing knowledge about the supported classes and attributes.

Best Practice:

Writing SPML schemas is considered a best practice, because it enables you to use general purpose tools (such as the OpenSPML Browser) to build requests.

Default SPML Configuration

The following example shows the default SPML configuration. The text of the SPML schema definitions have been omitted for brevity.

EXAMPLE 1–1 Default SPML Configuration

```
<Configuration name='SPML' authType='SPML'>
<Extension>
<Object>
  <Attribute name='classes'>
    <List>
      <Object name='person'>
        <Attribute name='type' value='User'/>
        <Attribute name='form' value='SPMLPerson'/>
        <Attribute name='default' value='true'/>
        <Attribute name='identifier' value='uid'/>
      </Object>
          <!-- Class 'user' defines no form so we'll default to a builtin simplified schema. I don't really like
          this but SimpleRpc currently depends on it. -->
      <Object name='user'>
        <Attribute name='type' value='User'/>
        <Attribute name='identifier' value='waveset.accountId'/>
      </Object>
          <!-- Class 'userview' defines the form "view" which causes the view to pass through
          unmodified-->       <Object name='userview'>
        <Attribute name='type' value='User'/>
        <Attribute name='form' value='view'/>
        <Attribute name='identifier' value='waveset.accountId'/>
        <Attribute name='multiValuedAttributes'>
          <List>
            <String>waveset.resources</String>
            <String>waveset.roles</String>
            <String>waveset.applications</String>
          </List>
        </Attribute>
      </Object>
      <Object name='role'>
        <Attribute name='type' value='Role'/>
        <Attribute name='form' value='SPMLRole'/>
        <Attribute name='default' value='true'/>
        <Attribute name='identifier' value='name'/>      <!-- attribute ...for now? -->
      </Object>
</Configuration>
```

Two classes are defined in this example:

- The standard `person` class
- An Identity Manager extension named `request`

The following attributes are supported in a class definition:

- `name` – Identifies the class name. The `name` value can correspond to an `<ObjectClassDefinition>` element in an SPML schema, although this value is not required. You can use this name as the value for the `objectclass` attribute in an `AddRequest` or a `SearchRequest`.
- `type` – Defines the Identity Manager view type used to manage instances of this class. Generally, this attribute is `User`, but it can be any repository type that is accessible through a view. For information about views, see *Sun Identity Manager Deployment Reference*.
- `form` – Identifies the name of a configuration object containing a form. This attribute contains the rules for transforming between the external attributes defined by the class and the internal view attributes.
- `default` – Specify `true` to indicate that this attribute is the default class for this type only. For more than one SPML class implemented on the same type, you must designate one class as the default.
- `identifier` – Each class typically defines one attribute as the object identity. The `identifier` attribute in the class definition specifies which attribute represents the identity. Where possible, use the `identifier` attribute value as the name of the corresponding repository object you create to represent the instance.
- `filter` – When evaluating an SPML search request for a class, you typically include all repository objects associated with that class in that search. This approach is fine for `User` objects, but some classes might be implemented by using generic types such as `TaskDefinition` or `Configuration`, not all of which are considered instances of the SPML class.

  To prevent unwanted objects from being included in the search, you can specify the `filter` attribute. The value is expected to be an `<AttributeCondition>` element or a `<List>` of `<AttributeCondition>` elements. Because custom classes are typically created for the `User` type, using a filter is uncommon. The default configuration uses filters to expose a subset of the `TaskInstance` objects that are known to have been created to handle asynchronous SPML requests.

Default Schemas

The `schemas` attribute contains a list of strings that contain the escaped XML for an SPML `<schema>` element. If you examine the `spml.xml` file, note that the schema elements are surrounded by a `CDATA`-marked section. Using `CDATA`-marked sections is convenient for escaping long strings of XML. When Identity Manager normalizes the `spml.xml` file, the `CDATA`-marked sections are converted into strings containing `&lt;` and `&gt;` character entities.

The default SPML configuration includes two schemas:

- A standard schema that is being defined by the SPML working group.
- A custom schema that is defined by Identity Manager. *Do not customize these schemas.*

  The Identity Manager schema contains a class definition for `request` and various extended requests for common account management operations.

`Configuration: SPMLPerson` Object

Each class defined in the `Configuration:SPML` object typically has an associated form object containing the rules for transforming between the external attribute model defined by the class and the internal model defined by the associated view.

The following example shows how the standard `person` class references a form.

EXAMPLE 1–2 Standard `Person` Class Referencing a Form

```
<Configuration name='SPMLPerson'>
   <Extension>
      <Form>
         <Field name='cn'>
            <Derivation><ref>global.fullname</ref></Derivation>
         </Field>
         <Field name='global.fullname'>
            <Expansion><ref>cn</ref></Expansion>
         </Field>
         <Field name='email'>
            <Derivation><ref>global.email</ref></Derivation>
         </Field>
         <Field name='global.email'>
            <Expansion><ref>email</ref></Expansion>
         </Field>
         <Field name='description'>
            <Derivation>
               <ref>accounts[Lighthouse].description</ref>
            </Derivation>
         </Field>
         <Field name='accounts[Lighthouse].description'>
            <Expansion><ref>description</ref></Expansion>
         </Field>
         <Field name='password'>
            <Derivation><ref>password.password</ref></Derivation>
         </Field>
         <Field name='password.password'>
            <Expansion><ref>password</ref></Expansion>
         </Field>
         <Field name='sn'>
```

EXAMPLE 1–2 Standard `Person` Class Referencing a Form *(Continued)*

```
            <Derivation><ref>global.lastname</ref></Derivation>
         </Field>
         <Field name='global.lastname'>
            <Expansion><ref>sn</ref></Expansion>
         </Field>
         <Field name='gn'>
            <Derivation><ref>global.firstname</ref></Derivation>
         </Field>
         <Field name='global.firstname'>
            <Expansion><ref>gn</ref></Expansion>
         </Field>
         <Field name='telephone'>
            <Derivation>
               <ref>accounts[Lighthouse].telephone</ref>
            </Derivation>
         </Field>
         <Field name='accounts[Lighthouse].telephone'>
            <Expansion><ref>telephone</ref></Expansion>
         </Field>
      </Form>
   </Extension>
</Configuration>
```

SPML class forms

- Contain no `<Display>` elements
- Are only defined for data transformation
- Are not intended for interactive editing

Each attribute in a class definition contains two field definitions:

- One field uses a `<Derivation>` expression to transform the internal view attribute `name` to the external `name`.
- One field uses an `<Expansion>` expression to transform the external `name` to the internal `name`.

The form is processed in such a way that when attributes are returned to the client, only the result of the `<Derivation>` expressions are included. When attributes are being sent from the client to the server, only `<Expansion>` expression results are assimilated back into the view. The effect is similar to the schema map of a Resource definition.

`Configuration: IDM Schema Configuration` Object

If you want to use attributes in an SPML search filter, you must define those attributes as *extended attributes* for Identity Manager users. Identity Manager stores extended attribute values in the repository, even when that value is also stored as a resource account attribute.

Try to minimize the use of extended attributes. Too many extended attributes can increase the repository size and might cause consistency problems between attributes stored in Identity Manager and the real value of the attribute stored on a resource. To use an attribute in an Identity Manager query, you must declare the attribute as an extended attribute to ensure that the value is accessible when the repository query indexes are built.

To include attributes in a user's set of summary attributes, you must define those attributes as extended attributes. You can use summary attributes to optimize searches by avoiding deserialization of the object XML, and instead return only a few of the most important user attributes. In the Identity Manager SPML implementation, summary attributes are returned when you do not explicitly provide a list of return attributes in the search request.

In the following example, `firstname`, `lastname`, `fullname`, `description`, and `telephone` are extended attributes that are present on the User `IDMObjectClassConfiguration` after being defined in `IDMAttributeConfigurations`. Only `firstname`, `lastname`, and `telephone` are queryable *and* summary attributes.

EXAMPLE 1–3 `telephone` and `description` Declared as Extended Attributes

```
<Configuration name="IDM Schema Configuration"
                  id='#ID#Configuration:IDM_Schema_Configuration'
                  authType='IDMSchemaConfig'>
       <IDMSchemaConfiguration>
           <IDMAttributeConfigurations>
                   <!-- this is the standard set -->
               <IDMAttributeConfiguration name='firstname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='lastname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='fullname'
                                          syntax='STRING'/>
                   <!-- these are the SPML extensions -->
               <IDMAttributeConfiguration name='description'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='telephone'
                                          syntax='STRING'/>
           </IDMAttributeConfigurations>
           <IDMObjectClassConfigurations>
               <IDMObjectClassConfiguration name='User'
                                            extends='Principal'
                                            description='User description'>
                   <IDMObjectClassAttributeConfiguration name='firstname'
                                                         queryable='true'
                                                         summary='true'/>
                   <IDMObjectClassAttributeConfiguration name='lastname'
                                                         queryable='true'
                                                         summary='true'/>
```

EXAMPLE 1–3 `telephone` and `description` Declared as Extended Attributes *(Continued)*

```
                <IDMObjectClassAttributeConfiguration name='fullname'/>
                <IDMObjectClassAttributeConfiguration name='description'/>
                <IDMObjectClassAttributeConfiguration name='telephone'
                                                      queryable='true'
                                                      summary='true'/>
            </IDMObjectClassConfiguration>
        </IDMObjectClassConfigurations>
    </IDMSchemaConfiguration>
</Configuration>
```

You can customize the list of attributes according to the needs of your site.

The names you choose for extended attributes depend on the mappings performed in the class form. Because the default `SPMLPerson` form maps `sn` into `lastname`, you must declare the extended attribute as `lastname`. The form does not transform the name of `telephone` or `description`, so the extended attribute name comes directly from the SPML schema.

Beyond declaring extended attributes, you must also modify the same `Configuration:` object to declare which of the attributes are queryable, or usable in an SPML filter, and which attributes are summary attributes to be returned by an optimized search result.

`TaskDefinition: SPMLRequest` Object

The `spml.xml` file also includes a brief definition for a new system task named `SpmlRequest`. You can use this task to implement asynchronous SPML requests. When the server receives an asynchronous request, it launches a new instance of this task and passes the SPML message as an input variable for the task. The server then returns the task instance repository ID in the SPML response for later status requests. For example:

```
<TaskDefinition name='SPMLRequest'
   executor='com.waveset.rpc.SpmlExecutor'
   execMode='asyncImmediate'
   resultLimit='86400'>
</TaskDefinition>
```

Do not change the definition name, the executor name, or the execution mode. You can change the `resultLimit` value. When asynchronous requests have completed, the system typically retains the result value for a specified time so the client can issue an SPML status request to obtain the results. How long to retain these results is site-specific.

Use a positive `resultLimit` value to specify how long (in seconds) the system can retain results after completing a task. The default value for `SPMLRequest` is typically 3600 seconds or approximately one hour. Other tasks default to 0 seconds unless you change the task name to a different value.

If negative, the request instance is never removed automatically.

Tip – To avoid cluttering the repository, set the `resultLimit` value to the shortest possible time.

Note – The Service Provider SPML interface does not support asynchronous requests.

Deployment Descriptor

You must edit the Identity Manager deployment descriptor, typically found in the `WEB-INF/web.xml` file, to contain a declaration for the servlet that receives SPML requests.

If you are having difficulty contacting the SPML web service, look in the `web.xml` file for a servlet declaration. The following example shows a servlet declaration.

EXAMPLE 1–4 Servlet Declaration

```
<servlet>
   <servlet-name>rpcrouter2</servlet-name>
   <display-name>OpenSPML SOAP Router</display-name>
   <description>no description</description>
   <servlet-class>
      org.openspml.server.SOAPRouter
   </servlet-class>
   <init-param>
      <param-name>handlers</param-name>
      <param-value>com.waveset.rpc.SimpleRpcHandler</param-value>
   </init-param>
   <init-param>
      <param-name>spmlHandler</param-name>
      <param-value>com.waveset.rpc.SpmlHandler</param-value>
   </init-param>
   <init-param>
      <param-name>rpcHandler</param-name>
      <param-value>com.waveset.rpc.RemoteSessionHandler</param-value>
   </init-param>
</servlet>
```

This declaration allows you to access `addRequest`, `modifyRequest`, and `searchRequest` web services through the URL:

`http://<host>:<port>/idm/servlet/rpcrouter2`

where

- *host* is the machine on which you are running Identity Manager.

- *port* is the number of the TCP port on which the server is listening.

Although you can, you are not required to define a *servlet-mapping*. Do not modify the contents of this servlet declaration.

Starting the OpenSPML Browser

You can use the OpenSPML Browser application to test your Identity Manager SPML configuration.

▼ To Start the OpenSPML Browser

1. **Open a command window.**
2. **At the command prompt, type:**

   ```
   lh spml
   ```

Connecting to the Identity Manager Server

▼ To Connect to the Identity Manager Server

1. **Open the OpenSPML browser and click the Connect tab.**

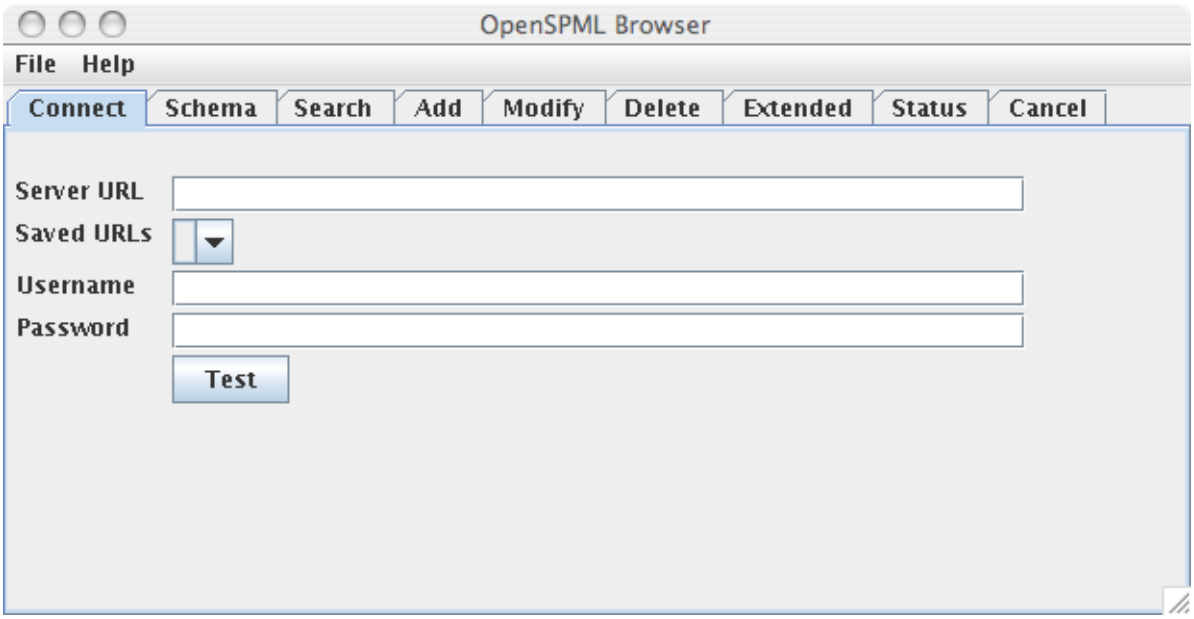

FIGURE 1–1 Example OpenSPML Browser

2 **Type the URL of the Identity Manager server into the Server URL field.**

For example, if the server is running on port `8080` on a local machine, the URL would be `http://`*host*`:8080/idm/servlet/rpcrouter2`.

Testing Your SPML Configuration

After connecting to the OpenSPML browser, use the following procedure to test your configuration.

▼ To Test Your SPML Configuration

1 **If necessary, click the Connect tab and click Test.**

A dialog displays to indicate that the connection was successful.

2 **Click the Schema tab and click Submit.**

The system displays a hierarchical view of the schemas supported by the Identity Manager server.

Troubleshooting If you cannot establish a successful connection, do the following:

- Verify that you typed the URL correctly.
- If the error message you receive contains phrases such as "`no response`" or "`connection refused`," the problem is most likely the host or port used in the connection URL.
- If the error message suggests that a connection was made, but the web application or servlet could not be located, the problem is most likely in the `WEB-INF/web.xml` file. See "Deployment Descriptor" on page 21 for more information.

Developing SPML Applications

After configuring the server, your SPML application requires a mechanism for sending SPML messages and receiving SPML responses. For Java™ applications, use the OpenSPML Toolkit to configure this mechanism.

Note – For optimal performance when you are working with the Identity Manager Web Service Interfaces, use the OpenSPML Toolkit that is co-packaged with Identity Manager.

Using the `openspml.jar` file from the `http://www.openspml.org/` web site might cause memory leaks.

The OpenSPML Toolkit provides the following components:

- Java class model for SPML messages
- Classes to send and receive messages on the client
- Classes to receive and process requests on the server

The following table describes the most important classes provided by the toolkit. Each request type has a corresponding class. Consult the Javadoc™ tool distributed with the toolkit for complete information.

TABLE 1–3 Classes Provided by the OpenSPML Toolkit

Class	Description
`AddRequest`	Constructs a message to request creation of a new object. You define the object type by passing an `objectclass` attribute. Other passed attributes must adhere to the schema associated with the object class. SPML does not yet define standard schemas, but you can configure Identity Manager to support almost all schemas.
`BatchRequest`	Constructs a message that can contain more than one SPML request.
`CancelRequest`	Constructs a message to cancel a request that was formerly executed asynchronously.

TABLE 1–3 Classes Provided by the OpenSPML Toolkit *(Continued)*

Class	Description
`DeleteRequest`	Constructs a message to request the deletion of an object.
`ModifyRequest`	Constructs a message to request modification of an object. Include only those attributes that you want to modify in the request. Attributes omitted from the request will retain their current values.
`SchemaRequest`	Constructs a message to request information about SPML object classes supported by the server.
`SearchRequest`	Constructs a message to request object attributes that match certain criteria.
`SpmlClient`	Presents a simple interface for sending and receiving SPML messages.
`SpmlResponse`	Includes the base class for objects representing response messages sent back from the server. Each request class has a corresponding response class, for example, `AddResponse` and `ModifyResponse`.
`StatusRequest`	Constructs a message to request the status of a request that was formerly executed asynchronously.

The Service Provider REF Kit contains an `SpmlUsage.java` file that demonstrates how to use the Service Provider SPML interface. This REF Kit also contains an `ant` script that compiles the `SpmlUsage` class.

Usage:

```
java [ -Dtrace=true ] com.sun.idm.idmx.example.SpmlUsage [ URL ]
```

where `URL` points to the Service Provider SPML interface. The URL defaults to

`http://`*host*`:`*port*`/idm/spespml`

where

- *host* is the machine on which you are running Identity Manager Service Provider.
- *port* is the number of the TCP port on which the server is listening.

You can enable trace for Service Provider to print Service Provider SPML messages to standard output.

`ExtendedRequest` Examples

The following table describes the different `ExtendedRequest` classes that you can use to send messages to and receive messages from the client.

TABLE 1–4 `ExtendedRequest` Classes for Sending and Receiving Messages

`ExtendedRequest` **Class**	**Description**
`changeUserPassword`	Constructs a message to request a user password change.
`deleteUser`	Constructs a message to request a user deletion.
`disableUser`	Constructs a message to request the disabling of a user.
`enableUser`	Constructs a message to request the enabling of a user.
`launchProcess`	Constructs a message to request the launch of a process.
`listResourceobjects`	Constructs a message to request the name of a resource object in the Identity Manager repository, and the type of object supported by that resource. The request returns a list of names.
`resetUserPassword`	Constructs a message to request the reset of a user password.
`runForm`	Allows you to create custom SPML requests that return information obtained by calling the Identity Manager Session API.

The server code converts each `ExtendedRequest` into a view operation.

The following examples illustrate the typical formats for an `ExtendedRequest` and its classes:

- "`ExtendedRequest` Example" on page 26
- "`deleteUser` Example" on page 27
- "`disableUser` Example" on page 27
- "`enableUser` Example" on page 27
- "`launchProcess` Example" on page 28
- "`listResourceObjects` Example" on page 28
- "`resetUserPassword` Example" on page 29
- "`runForm` Example" on page 29

`ExtendedRequest` Example

The following example shows the typical format for an `ExtendedRequest`.

EXAMPLE 1–5 `ExtendedRequest` Format

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "exampleuser");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Most SPML `ExtendedRequest` requests accept the following arguments:

- `accountId` – Identifies the Identity Manager user name.
- `accounts` – Presents resource names in a comma-delimited list.
  - If you pass an `accounts` argument, the specified SPML operation only updates the specified resources. You must include the `Lighthouse` attribute in a non-null accounts list if you want to update the Identity Manager user in addition to specific resource accounts.
  - If you do not pass an `accounts` argument, the operation updates all resource accounts linked to the user, including the Identity Manager user account.

`deleteUser` Example

The following example shows the typical format for a `deleteUser` request (View → Deprovision view).

Note – If you customize this request, there might be side effects.

EXAMPLE 1–6 `deleteUser` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("deleteUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

`disableUser` Example

The following example shows the typical format for a `disableUser` request (View → Disable view).

EXAMPLE 1–7 `disableUser` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("disableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

`enableUser` Example

The following example shows the typical format for an `enableUser` request (View → Enable view).

EXAMPLE 1–8 `enableUser` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("enableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

`launchProcess` Example

The following example shows the typical format for a `launchProcess` request (View → Process view).

EXAMPLE 1–9 `launchProcess` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("launchProcess");
req.setAttribute("process", "my custom process");
req.setAttribute("taskName", "my task instance");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

where:

- `launchProcess` – Starts the custom processes.
- `process` – Name of the `TaskDefinition` object in the Identity Manager repository to start.
- `taskName` – Name of the task needed to start the workflow.

  The `task instance` object holds the runtime state of the process.

The remaining attributes are arbitrary and they are passed into the task.

`listResourceObjects` Example

The following example shows the typical format for a `listResourceObjects` request.

EXAMPLE 1–10 `listResourceObjects` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("listResourceObjects");
req.setAttribute("resource", "LDAP");
req.setAttribute("type", "group");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

where:

- `resource` – Specifies the name of a resource object in the Identity Manager repository.
- `type` – Specifies the object type supported by that resource.

`resetUserPassword` Example

The following example shows the typical format for a `resetUserPassword` request (View → Reset User Password view).

EXAMPLE 1–11 `resetUserPassword` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("resetUserPassword");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

`runForm` Example

The following example shows the typical format for a `runForm` request.

EXAMPLE 1–12 `runForm` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("runForm");
req.setAttribute("form", "SPML Get Object Names");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

where `form` is the name of a configuration object containing a form.

Example Query Form

The following example shows a form that is used to run queries and return a list of the Role, Resource, and Organization names accessible to the current user.

EXAMPLE 1–13 Query Form

```
<Configuration name='SPML Get Object Names'>
  <Extension>
    <Form>
      <Field name='roles'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Role</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='resources'>
        <Derivation>
```

EXAMPLE 1–13 Query Form *(Continued)*

```
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Resource</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='organizations'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>ObjectGroup</s>
          </invoke>
        </Derivation>
      </Field>
    </Form>
  </Extension>
</Configuration>
```

You use the `runForm` request to create custom SPML requests that return information obtained by calling the Identity Manager Session API. For example, when configuring a user interface for editing user accounts, you might want to provide a selector that displays the names of the organizations, roles, resources, and policies that can be assigned to a user.

You can configure the SPML interface to expose these objects as SPML object classes and use `searchRequest` to query for their names. However, this configuration requires four `searchRequest` requests to gather the information. To reduce the number of SPML requests, encode the queries in a form by using a single `runForm` request to perform the queries and return the combined results.

Using Trace With SPML

SPML includes options for turning on trace output so you can log Identity Manager SPML traffic and diagnose problems.

For more information about tracing SPML, see Chapter 5, "Tracing and Troubleshooting," in *Sun Identity Manager 8.1 System Administrator's Guide*.

Example Methods for Implementing SPML

The following examples illustrate some common methods for implementing SPML.

`AddRequest` Method

The following example shows a typical `AddRequest` method.

EXAMPLE 1–14 `AddRequest` Example

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/idm/spml");
   AddRequest req = new AddRequest();
   req.setObjectClass("person");
   req.setIdentifier("maurelius");
   req.setAttribute("gn", "Marcus");
   req.setAttribute("sn", "Aurelius");
   req.setAttribute("email", "maurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully created");
```

`ModifyRequest` Method

This section contains two authenticated SPML `ModifyRequest` examples.

EXAMPLE 1–15 Authenticated SPML Request Example

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/idm/spml");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.addModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

EXAMPLE 1–16 Authenticated SPML Request Example With `LighthouseClient`

```
LighthouseClient client = new LighthouseClient();
   client.setURL("http://example.com:8080/idm/spml");
   client.setUser("maurelius");
   client.setPassword("xyzzy");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.addModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

The only difference between these examples is that the second example uses the `LighthouseClient` class and two additional method calls to `client.setUser` and `client.setPassword`. For example, you could use this example to avoid setting a proxy user in `Waveset.properties`, which results in the audit log reflecting the specified user instead of the proxy user.

This example is authenticated by `client.setUser` and `client.setPassword` when the request is sent.

`SearchRequest` Method

The following example shows a typical `SearchRequest` method.

EXAMPLE 1–17 `SearchRequest` Example

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/idm/spml");
   SearchRequst req = new SearchRequest();
   // specify the attributes to return
   req.addAttribute("sn");
   req.addAttribute("email");
   // specify the filter
   FilterTerm ft = new FilterTerm();
   ft.setOperation(FilterTerm.OP_EQUAL);
   ft.setName("gn");
   ft.setValue("Jeff");
   req.addFilter(ft);
   SearchResponse res = (SearchResponse)client.request(req);
   // display the results
   List results = res.getResults();
   if (results != null)  {
      for (int i = 0 ; i < results.size() ; i++) {
```

EXAMPLE 1–17 `SearchRequest` Example *(Continued)*

```
        SearchResult sr = (SearchResult)results.get(i);
        System.out.println("Identifier=" +
                           sr.getIdentifierString() +
                           " sn=" +
                           sr.getAttribute("sn") +
                           " email=" +
                           sr.getAttribute("email"));
        }
    }
```

◆ ◆ ◆ CHAPTER 2

Using SPML 2.0 With Sun Identity Manager Web Services

This chapter describes SPML 2.0 support in Identity Manager, including which features are supported and why, how to configure SPML 2.0 support, and how to extend support in the field.

This chapter covers the following topics:

Note – The concepts in this chapter *relate specifically* to SPML 2.0. Unless noted otherwise, all references to SPML in this chapter indicate Version *2.0*.

You should also read Chapter 1, "Using SPML 1.0 With Sun Identity Manager Web Services," for a description of some basic concepts related to using SPML.

Important Notes About Using SPML 2.0

Before you start working with SPML 2.0 and Identity Manager Web Services, note the following:

- For optimal performance when you are working with the Identity Manager Web Service interfaces, use the OpenSPML Toolkit that is co-packaged with Identity Manager. Using the `openspml.jar` file from the `http://www.openspml.org/` web site might cause memory leaks.
- When implementing SPML 2.0, you must modify the configuration to add the `spml2ObjectClass` attribute to your schema. The `objectclass` attribute value provided in previous releases is now maintained in the `spml2ObjectClass` attribute.

- You cannot access Sun Identity Manager Service Provider (Service Provider) features through SPML 2.0. These features are only available through SPML Version 1.0. See Chapter 1, "Using SPML 1.0 With Sun Identity Manager Web Services," for more information.

Basic SPML 2.0 Concepts

This section explains some basic concepts about SPML 2.0:

How SPML 2.0 Compares to SPML 1.0

Identity Manager Web Services support both SPML Version 1.0 and Version 2.0 protocols (open standards for service provisioning using XML) for communication with provisioning systems.

Note – See Chapter 1, "Using SPML 1.0 With Sun Identity Manager Web Services," for information about using SPML Version 1.0.

SPML 2.0 offers many improvements over SPML 1.0, including the following:

- Where SPML 1.0 has been called a slightly improved Directory Services Markup Language (DSML), SPML 2.0 defines an extensible protocol (through *Capabilities*) with support for a DSML profile, as well as *XML Schema* profiles. SPML 2.0 differentiates between the protocol and the data it carries.
- The SPML 2.0 protocol enables better interoperability between vendors, especially for the `Core` capabilities (those found in 1.0).

  You can "extend" SPML 1.0 using `ExtendedRequest`, but there is no guidance about what those requests can be. SPML 2.0 defines a set of "standard capabilities" that allow you to add support in well-defined ways.
- SPML 2.0 provides additional capabilities (see Table 2–1) that enable you to extend capabilities or add new capabilities in the future.

TABLE 2–1 SPML Capabilities

SPML 1.0 Capabilities	SPML 2.0 Capabilities
`Add`	`Add`
`Modify`	`Modify`
`Delete`	`Delete`
`Lookup`	`Lookup`
`SchemaRequest`	`ListTargets`
`Search`	`Search`
`ExtendedRequest`	Captured in "standard" capabilities: ■ `Async` – Process requirements asynchronously ■ `Batch` – Process a batch of requests ■ `Bulk` – Process `modifies` or `deletes` using iteration ■ `Password` – Change, set, reset, validate, or expire passwords ■ `Reference` – Refer to PSOs between targets ■ `Suspend` – Enable or disable PSOs ■ `Update` – Find change records for objects that have been updated (can also be captured in "custom" capabilities)

How SPML 2.0 Concepts Are Mapped to Identity Manager

SPML 2.0 uses its own terminology to discuss the objects that are managed by a provisioning system.

Note – See the OASIS SPML 2.0 specifications at `http://www.openspml.org/`.

The following sections describe how SPML 2.0 concepts are mapped into Identity Manager:

Understanding Targets

A *target* is a logical end point in the server. Every target is named and declares the schema of the objects that it manages. Targets also declare which capabilities (a set of requests) are supported. See "Understanding PSOs" on page 38 for more information.

Currently, Identity Manager supports only *one* target. You cannot declare multiple targets. You can name this target anything you want, but the data objects' format must conform to the DSML profile.

A supported target is the one target defined in the `spml2.xml` file (`Configuration:SPML2` object). For example, in "`ListTargetsRequest` Examples" on page 46, `ListTargetResponse` returns one target, `spml2-DSML-Target`.

Understanding PSOs

As mentioned in the previous section, targets manage Provisioning Service Objects (PSOs). A *PSO* is somewhat analogous to a *view* in Identity Manager, but without behavior. Consequently, you can think of a PSO as the data portion of an Identity Manager view, a User view in particular.

Note – Identity Manager only manages Users and requires you to define a user extended attribute called `spml2ObjectClass`.

For Identity Manager's purposes, a PSO is a collection of attributes that are mapped (using a form) to and from a User view. Each object specifies an `objectclass` attribute that is used to map the object to an `objectclass` definition in the schema defined for the target.

The `objectclass` attribute is used to find the following:

- A `repoType` that is provided to support additional targets later
- A form that maps the attributes to and from the Identity Manager view

Understanding PSOIdentifiers

SPML includes an object ID that is called a PSOIdentifier (*PsoID*).

OASIS SPML 2.0 specifications recommend that PsoIDs be opaque to a requestor (client). Consequently, Identity Manager uses repository IDs (*repoIDs*) as PsoIDs when adding PSOs to the system.

A repoID is distinct and it is not meant for presentation to a user. When displaying a PSO to a user, the requestor should use the equivalent of the `waveset.accountid` or whatever attributes are used in the Identity template to present the object's ID.

When identifying the PSO (as in a `ModifyRequest`), the requestor should use the repoID and not the `waveset.accountId`. Although the requestor can use the `waveset.accountId` as a PsoID, doing so is not recommended and it might change in a future release. Requestors should try to keep the PsoID opaque.

PSOs use an `objectclass` attribute to specify the object type. If this attribute is not present when a request is made, Identity Manager allows you to specify and use a "default" object class, such as `SPMLUser`. Internally, the `objectclass` value is maintained as an `spml2ObjectClass` attribute for users. For Identity Manager this attribute must be a user extended attribute. You might not see an `spml2ObjectClass` attribute for users that existed before you enabled SPML 2.0.

Understanding Open Content and `OperationalAttributes`

SPML makes heavy use of `xsd:any` elements in the `.xsds` file to provide what the specification refers to as *Open Content*. In SPML, Open Content means that most elements can contain elements of any type. Identity Manager uses this idea to provide *OperationalNVPs* (`NameValuePairs`) and *OperationalAttributes* that control processing. `OperationalNVPs` appear as elements in the XML, while `OperationalAttributes` appear as attributes. See the OpenSPML 2.0 Toolkit at `http://www.openspml.org` for more information.

You can use one NVP in all requests *except* `ListTargetsRequests`, and in *all* responses. Identity Manager stores a `sessionToken` in an `OperationalNVPs` called `session` that allows the system to cache sessions on behalf of the user and improves efficiency. For more information about `OperationalNVPs` and `OperationalAttributes`, see "Supported SPML 2.0 Capabilities" on page 39.

Supported SPML 2.0 Capabilities

Identity Manager supports all `Core` capabilities in the SPML 2.0 specification that use the DSML profile. Identity Manager also supports some of the optional standard capabilities (such as `Batch` and `Async`) and partially supports some standard capabilities (such as `Bulk`).

This section describes which SPML 2.0 capabilities are supported in Identity Manager (where Identity Manager knowingly varies from the specification and profile documents) and which `OperationalAttributes` are required by Identity Manager.

The topics in this section include the following:

- "Search Capability" on page 53

Note – Identity Manager does not support the `Reference` capability, the `Updates` capability, or the `CapabilityData` class.

The `CapabilityData` class is used to implement custom capabilities. Identity Manager does not support the `CapabilityData` class because none of the supported capabilities use this class.

The OpenSPML 2.0 Toolkit supports `CapabilityData` in the marshallers, unmarshallers, and so forth.

`Core` Capabilities

Identity Manager supports the `Core` capabilities described in the following table.

TABLE 2–2 `Core` Capabilities

Capability	Description	Caveats
`AddRequest`	Adds a specified PSO to the system.	Identity Manager officially supports only a single target.
`DeleteRequest`	Deletes a specified PSO from the system.	Identity Manager officially supports only a single target.
`ListTargetsRequest`	Lists the targets that are available through Identity Manager.	■ Identity Manager officially supports a single target. ■ Identity Manager does not require you to use `ListTargets` as the first call in a conversation. However, it does allow `operationalAttributes` on this request to specify a username/password pair for establishing a session with the server. (You can also use `Waveset.properties`.) In general, it is more efficient to log in and use the session token. Identity Manager provides a class called `SessionAwareSpml2Client` for this purpose.
`LookupRequest`	Finds and returns the attributes of the named PSO.	None
`ModifyRequest`	Modifies specified PSO attributes.	Due to a discrepancy between the main SPML 2.0 specification and the DSML Profile specification, Identity Manager *does not* support `select`, `component`, and so forth. Instead Identity Manager uses DSML Modification Mode and elements according to the DSML Profile.

Note –

General caveats include the following:

- You can provide `username` and `password` values for the `ListTargetsRequest` request. These values are used as credentials to establish a `session`, which is identified by the `session token` value returned in the`ListTargetsRequest` response. This session is the context for all following requests that include that `session token` value as an operational attribute.

  Another way to set up the session is to provide values for the `soap.username` and `soap.password` attributes in `Waveset.properties`. In this case, no `session token` is required.
- Identity Manager supports only the DSML Profile.

`AddRequest` and `ListTargetsRequest` examples follow.

Core Capability Examples

This section provides several Java, XML, and JSP examples.

The following examples adds a user with several attributes. The first example returns all data, while the second returns only the identifier.

EXAMPLE 2–1 Example `AddRequest`

```
// ReturnData.EVERYTHING example
SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080/
idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

AddRequest req = new AddRequest();
req.setReturnData(ReturnData.EVERYTHING);
Extensible attrs = new Extensible();
attrs.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
attrs.addOpenContentElement(new DSMLAttr("accountId", "sempiricus"));
attrs.addOpenContentElement(new DSMLAttr("credentials", "password"));
attrs.addOpenContentElement(new DSMLAttr("firstname", "Sextus"));
attrs.addOpenContentElement(new DSMLAttr("lastname", "Empiricus"));
req.setData(attrs);

AddResponse res = (AddResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Received positive add response.");
}

PSO pso = res.getPso();
System.out.println("PSO ID: " + pso.getPsoID().getID());
Extensible psoData = pso.getData();
for (OpenContentElement oce : psoData.getOpenContentElements()) {
```

EXAMPLE 2–1 Example AddRequest *(Continued)*

```
    if (oce instanceof DSMLAttr) {
        DSMLAttr attr = (DSMLAttr) oce;
        System.out.println(attr.getName() + ": " + attr.getValues()[0].getValue());
    }
}

// ReturnData.IDENTIFIER example
SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080/
idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

AddRequest req = new AddRequest();
req.setReturnData(ReturnData.IDENTIFIER);
Extensible attrs = new Extensible();
attrs.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
attrs.addOpenContentElement(new DSMLAttr("accountId", "catullus"));
attrs.addOpenContentElement(new DSMLAttr("credentials", "password"));
attrs.addOpenContentElement(new DSMLAttr("firstname", "Gaius"));
attrs.addOpenContentElement(new DSMLAttr("lastname", "Catullus"));
req.setData(attrs);

AddResponse res = (AddResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Received positive add response.");
}

PSO pso = res.getPso();
System.out.println("PSO ID: " + pso.getPsoID().getID());
Extensible psoData = pso.getData();
if (psoData == null) {
    System.out.println("PSO contains no data, as expected.");
}
```

The following example shows an account lookup.

EXAMPLE 2–2 Example LookupRequest

```
// Lookup example
SessionAwareSpml2Client client = new
     SessionAwareSpml2Client("http://example.com:8080/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

PSOIdentifier psoId = new PSOIdentifier("maurelius", null, null);

LookupRequest req = new LookupRequest();
```

EXAMPLE 2–2 Example `LookupRequest` *(Continued)*

```
req.setPsoID(psoId);
req.setExecutionMode(ExecutionMode.SYNCHRONOUS);

try {
    LookupResponse res = (LookupResponse) client.send(req);
    if (res.getStatus().equals(StatusCode.SUCCESS)) {
        System.out.println("Performed account lookup.");
    }
    PSO pso = res.getPso();
} catch (Spml2ExceptionWithResponse e) {
    System.out.println("Lookup failed: " + e.getMessage());
    LookupResponse res = (LookupResponse) e.getResponse();
}
```

The following example changes the `lastname` parameter to `Antoninus`.

EXAMPLE 2–3 Example `ModifyRequest`

```
SessionAwareSpml2Client client = new
     SessionAwareSpml2Client("http://example.com:8080/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

PSOIdentifier psoId = new PSOIdentifier("maurelius", null, null);

ModifyRequest req = new ModifyRequest();
req.setPsoID(psoId);
Modification modification = new Modification();
modification.addOpenContentElement(new DSMLModification("lastname", "Antoninus",
                                                        ModificationMode.REPLACE));
req.addModification(modification);

ModifyResponse res = (ModifyResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Modified account.");
}
```

The following example shows the SPML 2.0 request that was sent.

EXAMPLE 2–4 Example Request XML

```
<addRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid-spmlv2'
executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml
    name='session' value='AAALPgAAYD0A...'/>
  <data>
```

EXAMPLE 2–4 Example Request XML *(Continued)*

```
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
      <dsml:value>exampleSpml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
      <dsml:value>spml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
      <dsml:value>pwdpwd</dsml:value>
    </dsml:attr>
  </data>
</addRequest>
```

This example shows the body of the SPML response that was returned to the client.

EXAMPLE 2–5 Example Response XML

```
<addResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='success'
requestID='rid-spmlv2'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='session' value='AAALPgAAYD0A...'/>
  <pso>
    <psoID ID='anSpml2Person'/>
    <data>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
        <dsml:value>anSpml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
        <dsml:value>spml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
        <dsml:value>pwdpwd</dsml:value>
      </dsml:attr>
    </data>
  </pso>
</addResponse>
```

EXAMPLE 2–6 Example JSP

The following example consists of a `.jsp` file that invokes an `AddRequest` through Identity Manager's `SessionAwareSpml2Client` class.

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%>
```

EXAMPLE 2–6 Example JSP *(Continued)*

```
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>

<%
final String url = "http://host:port/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
<body>
<%

 // need a client.
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // login
 client.login("configurator", "password");

 // AddRequest
 String rid = "rid-spmlv2"; // The RequestId is not strictly required.

 Extensible data = new Extensible();
 data.addOpenContentElement(new DSMLAttr("accountId", user));
 data.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
 data.addOpenContentElement(new DSMLAttr("credentials", password));

 AddRequest add = new AddRequest(rid, // String requestId,
                ExecutionMode.SYNCHRONOUS, // ExecutionMode executionMode,
                null, // PSOIdentifier type,
                null, // PSOIdentifier containerID,
                data, // Extensible data,
                null, // CapabilityData[] capabilityData,
                null,  // String targetId,
                null  // ReturnData returnData
            );

    // Submit the request
    Response res = client.send( add );
%>
<%= res.toString()%>
</body>
</html>
```

`ListTargetsRequest` Examples

The examples in this section illustrate the `ListsTargetsRequest` capabilities that are available using Identity Manager.

EXAMPLE 2–7 Example Client Code

The following example shows how a `.jsp` file invokes a `ListTargetsRequest` through Identity Manager's `SessionAwareSpml2Client` class.

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%>
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>
<%
final String url = "http://host:port/idm/servlet/openspml2";
%>
<html>
<head><title>SPML2 Test</title></head>
<body>
<%
 // need a client.
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );
 // login (sends a ListTargetsRequest)
 Response res = client.login("configurator", "password");
%>
<%= res.toString()%>
</body>
</html>
```

EXAMPLE 2–8 Example Request XML

This next example shows the body of the SPML request that is sent.

```
<listTargetsRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid[7013]'
    executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='accountId' value='configurator'/>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='password' value='password'/>
</listTargetsRequest>
```

This example shows the body of the SPML response that is received by or returned to the client.

EXAMPLE 2–9 Example Response XML

```
   <openspml:operationalNameValuePair
xmlns:openspml="urn:org:openspml:v2:util:xml" name="session" value="AAAM+wAAaC..."/>
   <target targetID="spml2-DSML-Target" profile="urn:oasis:names:tc:SPML:2:0:DSML">
      <schema>
         <spmldsml:schema xmlns:spmldsml="urn:oasis:names:tc:SPML:2:0:DSML">
            <spmldsml:objectClassDefinition name="spml2Person">
               <spmldsml:memberAttributes>
                  <spmldsml:attributeDefinitionReference required="true"
name="objectclass"/>
                  <spmldsml:attributeDefinitionReference required="true"
name="accountId"/>
                  <spmldsml:attributeDefinitionReference required="true"
name="credentials"/>
                  <spmldsml:attributeDefinitionReference name="firstname"/>
                  <spmldsml:attributeDefinitionReference name="lastname"/>
                  <spmldsml:attributeDefinitionReference name="emailAddress"/>
               </spmldsml:memberAttributes>
            </spmldsml:objectClassDefinition>
            <spmldsml:attributeDefinition name="objectclass"/>
            <spmldsml:attributeDefinition description="Account Id" name="accountId"/>
            <spmldsml:attributeDefinition description="Credentials, e.g. password"
name="credentials"/>
            <spmldsml:attributeDefinition description="First Name" name="firstname"/>
            <spmldsml:attributeDefinition description="Last Name" name="lastname"/>
            <spmldsml:attributeDefinition description="Email Address"
name="emailAddress"/>
         </spmldsml:schema>
         <supportedSchemaEntity entityName="spml2Person"/>
      </schema>
      <capabilities>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:async"/>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:batch"/>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:bulk"/>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:password"/>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:suspend"/>
         <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:search"/>
      </capabilities>
   </target>
</listTargetsResponse>
```

Async Capabilities

Identity Manager supports the Async capabilities described in the following table.

TABLE 2–3 Async Capabilities

Capability	Description	OperationalAttributes
CancelRequest	Cancels a request, using the request ID.	None
StatusRequest	Returns the status of a request, using the request ID.	None

The following example performs an asynchronous status request.

EXAMPLE 2–10 Example Async Requests

```
String REQUEST_ID = "test-req-id-00000000000001";
String PRE_ASYNC_DELAY = "45";
String POST_ASYNC_DELAY = "15";

SessionAwareSpml2Client client = new
   SessionAwareSpml2Client("http://example.com:8080/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

PSOIdentifier psoId = new PSOIdentifier("maurelius", null, null);
LookupRequest lookupReq = new LookupRequest();
lookupReq.setPsoID(psoId);
lookupReq.setReturnData(ReturnData.EVERYTHING);
lookupReq.setRequestID(REQUEST_ID);
lookupReq.setExecutionMode(ExecutionMode.ASYNCHRONOUS);

lookupReq.addOpenContentElement(
    new OperationalNameValuePair(
        com.sun.idm.rpc.spml2.async.AsyncExecutorTask.IDM_BEFORE_ASYNC_DELAY_NAME,
        PRE_ASYNC_DELAY
    )
);
lookupReq.addOpenContentElement(
    new OperationalNameValuePair(
        com.sun.idm.rpc.spml2.async.AsyncExecutorTask.IDM_AFTER_ASYNC_DELAY_NAME,
        POST_ASYNC_DELAY
    )
);

LookupResponse lookupRes = (LookupResponse) client.send(lookupReq);
if (lookupRes.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Lookup response is pending.");
}

StatusRequest statusReq = new StatusRequest();
statusReq.setAsyncRequestID(REQUEST_ID);
```

EXAMPLE 2–10 Example Async Requests *(Continued)*

```
statusReq.setReturnResults(false);
statusReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);
StatusResponse statusRes = (StatusResponse) client.send(statusReq);
if (lookupRes.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Status response received.");
}

java.util.Iterator responseIterator = statusRes.responseIterator();
while (responseIterator.hasNext()) {
    Response nestedResponse = (Response) responseIterator.next();
    if (nestedResponse instanceof LookupResponse) {
        if (nestedResponse.getStatus().equals(StatusCode.SUCCESS)) {
            System.out.println("Successfully received nested lookup response.");
        }
    }
}
```

`Batch` Capability

Identity Manager supports the `Batch` capability described in the following table.

TABLE 2–4 `Batch` Capability

Capability	Description	`OperationalAttributes`
`BatchRequest`	Executes a batch of requests.	None

The following example performs a batch add operation.

EXAMPLE 2–11 Example Batch Request

```
SessionAwareSpml2Client client = new
    SessionAwareSpml2Client("http://example.com:8080/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

BatchRequest batchReq = new BatchRequest();
for (int i = 1; i <= 10; i++) {
    AddRequest addReq = new AddRequest();
    Extensible attrs = new Extensible();
    attrs.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
    attrs.addOpenContentElement(new DSMLAttr("accountId", "test_" +
       String.format("%03d", i)));
    attrs.addOpenContentElement(new DSMLAttr("credentials", "password"));
    addReq.setData(attrs);
    addReq.setReturnData(ReturnData.EVERYTHING);
    batchReq.addRequest(addReq);
```

EXAMPLE 2–11 Example Batch Request *(Continued)*

```
}
BatchResponse batchRes = (BatchResponse) client.send(batchReq);
if (batchRes.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Successfully performed batch add operation.");
}
```

`Bulk` Capabilities

Identity Manager supports the `Bulk` capabilities described in the following table.

TABLE 2–5 `Bulk` Capabilities

Capability	Description	`OperationalAttributes`
`BulkDeleteRequest`	Executes a bulk delete of PSOs.	None
`BulkModifyRequest`	Executes a bulk modify of matching PSOs.	None

EXAMPLE 2–12 Example Bulk Delete Request

The following example bulk deletes all users with the lastname *Aurelius*.

```
SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080/
idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

BulkDeleteRequest req = new BulkDeleteRequest();

Substrings term = new Substrings();
term.setName("lastname");
term.setInitial(new DSMLValue("Aurelius"));
Filter filter = new Filter(term);
Query q = new Query();
q.setScope(Scope.ONELEVEL);
q.addQueryClause(filter);

req.setExecutionMode(ExecutionMode.SYNCHRONOUS);
req.setQuery(q);

BulkDeleteResponse res = (BulkDeleteResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Successfully performed bulk delete operation.");
}
```

`Password` Capabilities

Identity Manager supports the `Password` capabilities described in the following table.

TABLE 2–6 `Password` Capabilities

Capability	Description	Caveats
`ExpirePasswordRequest`	Expires a password.	■ You cannot specify resources or targets. Doing so causes the Identity Manager `User` object password to expire; which then causes the password on all of the user's resources to expire. ■ Identity Manager does not support the `remainingLogins` attribute. If you set this attribute to anything other than the default (1 or less), an `OperationNotSupported` error occurs.
`ResetPasswordRequest`	Resets a password and returns the new value on all accounts.	Passwords are sensitive. Use SSL or some other secure transport.
`SetPasswordRequest`	Sets a password.	Passwords are sensitive. Use SSL or some other secure transport.
`ValidatePasswordRequest`	Determines whether a given password is valid.	Passwords are sensitive. Use SSL or some other secure transport.

Example `Password` capabilities follow.

`ResetPasswordRequest` Example

The following example illustrates a typical `ResetPasswordRequest`.

EXAMPLE 2–13 Example `ResetPasswordRequest`

```
ResetPasswordRequest rpr = new ResetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
 rpr.setPsoID(psoId);
...
```

`SetPasswordRequest` Example

The following example illustrates a typical `SetPasswordRequest`.

EXAMPLE 2–14 Example `SetPasswordRequest`

```
SetPasswordRequest spr = new SetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
```

EXAMPLE 2–14 Example `SetPasswordRequest` *(Continued)*

```
spr.setPsoID(psoId);
spr.setPassword("newpassword");
spr.setCurrentPassword("oldpassword");
...
```

`ValidatePasswordRequest` Example

The following example illustrates a typical `ValidatePasswordRequest`.

EXAMPLE 2–15 Example `ValidatePasswordRequest`

```
ValidatePasswordRequest vpr = new ValidatePasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
vpr.setPsoID(psoId);
vpr.setPassword("apassword");
...
```

`Suspend` Capabilities

Identity Manager supports the `Suspend` capabilities described in the following table.

TABLE 2–7 `Suspend` Capabilities

Capability	Description	Caveats
`ActiveRequest`	Returns a boolean value that indicates whether the user is enabled.	None
`ResumeRequest`	Resumes (enables) a PSO user.	Does not support `EffectiveDate`. If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error.
`SuspendRequest`	Suspends an account/PSO (disables).	Does not support `EffectiveDate`. If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error.

EXAMPLE 2–16 Suspend Examples

The following examples suspend and resume a request.

```
// Disable example
SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080
/idm/servlet/openspml2");
```

```
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

PSOIdentifier psoId = new PSOIdentifier("maurelius", null, null);
SuspendRequest req = new SuspendRequest();
req.setPsoID(psoId);

SuspendResponse res = (SuspendResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Account successfully disabled.");
}

// Enable example
SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080
/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

PSOIdentifier psoId = new PSOIdentifier("maurelius", null, null);
ResumeRequest req = new ResumeRequest();
req.setPsoID(psoId);

ResumeResponse res = (ResumeResponse) client.send(req);
if (res.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Account successfully enabled.");
}
```

Search Capability

Identity Manager supports the Search capabilities described in the following table.

TABLE 2–8 `Search` Capabilities

Capability	Description	`OperationalAttributes`
`SearchRequest`	Returns as many users as it can find that match the specified criteria within the allotted amount of time. If additional users exist, the request also returns a handle that can be used within an *IterateRequest*.	None
`IterateRequest`	Continues an existing incomplete SearchRequest or *IterateRequest*.	None
CloseIteratorRequest	Ends an *IterateRequest*. You may terminate an *IterateRequest* before it terminates.	None

Search Filter Configuration

The DSML search filter is translated to a set of repository attribute conditions. A necessary part of the translation involves mapping the external DSML attribute name to a queryable repository attribute. A set of attribute mappings for each searchable type must be defined in the configuration.

The filter is analyzed and processed according to one of the following categories:

- Fully-indexed. The filter is converted to equivalent attribute conditions that find exactly the required objects from the repository. In this case, there is no need to further filter the resulting objects. A fully indexed search is much faster than a partially indexed or unindexed search because it does not instantiate views of the queried objects.
- Partially-indexed. At least one attribute condition is derived from the filter that finds a superset of the required objects from the repository. For each object, the system instantiates a view of the object that contains, at minimum, those attributes specified in the filter. It also checks whether the object matches the filter.
- Unindexed. No attribute conditions can be derived from the filter to apply against the repository. This type is similar to a partially-indexed search, except that all objects in the repository of the requested type are processed.
- Unsupported. The filter cannot be evaluated. The search operation is rejected.

The SPML2 configuration object contains the following properties

- `SearchIndexRequired`. Specifies whether partially-indexed or unindexed searches are accepted. An example of a search that is not fully-indexed is one whose filter specifies a non-queryable attribute. The possible values are as follows:
  - `full`. Indicates that only fully indexed search operations are accepted. This is the default value.
  - `partial`. Indicates that partial or fully indexed search operations are accepted.
  - `none`. Indicates that unindexed search operations are accepted, in addition to partial and fully indexed operations.
- `SearchSizeLimit`. An integer value specifying the maximum total number of objects that can be returned from a search request. If the search creates an iterator, all subsequent iterator responses are counted toward the total. A value of 0 indicates there is no limit.

  When the limit is exceeded, results are returned up to the limit, and an informational message is returned to the client indicating that the limit was exceeded.
- `SearchTimeLimit`. An integer value specifying the maximum time, in seconds, that an individual SPMLv2 search request or iterator request is allowed to take before a response is returned. A value of 0 means there is no limit.
- `queryMappings`. A map of existing SPML attribute names and Identity Manager repository attribute names.

- `IteratorTimeout`. An integer value specifying the maximum time, in seconds, that an SPMLv2 iterator remains valid after it was last provided in a response. A value must be specified.

The following capabilities are not supported:

- `org.openspml.v2.profiles.dsml.ApproxMatch`
- `org.openspml.v2.profiles.dsml.ExtensibleMatch`

Search Examples

The following examples illustrate how to perform searches.

EXAMPLE 2–17 SearchRequest Example

```
ArrayList<PSO> psos = new ArrayList<PSO>();

SessionAwareSpml2Client client = new SessionAwareSpml2Client("http://example.com:8080
/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

SearchRequest searchReq = new SearchRequest();
EqualityMatch acctIdTerm = new EqualityMatch("firstname", new DSMLValue("Marcus"));
Present emailTerm = new Present("emailAddress");
org.openspml.v2.profiles.dsml.And terms = new
    org.openspml.v2.profiles.dsml.And(new FilterItem[] { acctIdTerm, emailTerm });
Filter filter = new Filter(terms);
Query q = new Query();
q.setScope(Scope.ONELEVEL);
q.addQueryClause(filter);

searchReq.setQuery(q);
searchReq.setReturnData(ReturnData.IDENTIFIER);
searchReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);

SearchResponse searchRes = (SearchResponse) client.send(searchReq);
if (searchRes.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Received search response.");
    for (PSO pso : searchRes.getPSOs()) {
        psos.add(pso);
    }

    ResultsIterator iterator = searchRes.getIterator();

    while (iterator != null) {
        IterateRequest iterReq = new IterateRequest();
        iterReq.setIterator(iterator);
        iterReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);
```

EXAMPLE 2–17 SearchRequest Example *(Continued)*

```
        IterateResponse iterRes = (IterateResponse) client.send(iterReq);
        if (iterRes.getStatus().equals(StatusCode.SUCCESS)) {
            System.out.println("Found an iterator.");
        }
        for (PSO pso : iterRes.getPSOs()) {
            psos.add(pso);
        }
        iterator = iterRes.getIterator();
    }
}
```

EXAMPLE 2–18 CloseIterator Example

```
// Close iterator example
ArrayList<PSO> psos = new ArrayList<PSO>();

SessionAwareSpml2Client client = new
SessionAwareSpml2Client("http://example.com:8080/idm/servlet/openspml2");
ListTargetsResponse loginInfo = client.login("Configurator", "configurator");

SearchRequest searchReq = new SearchRequest();

Present term = new Present("emailAddress");
Filter filter = new Filter(term);
Query q = new Query();
q.setScope(Scope.ONELEVEL);
q.addQueryClause(filter);

searchReq.setQuery(q);
searchReq.setReturnData(ReturnData.EVERYTHING);
searchReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);

SearchResponse searchRes = (SearchResponse) client.send(searchReq);
if (searchRes.getStatus().equals(StatusCode.SUCCESS)) {
    System.out.println("Received search response.");
    for (PSO pso : searchRes.getPSOs()) {
        psos.add(pso);
    }

    ResultsIterator iterator = searchRes.getIterator();

    while (iterator != null) {
        IterateRequest iterReq = new IterateRequest();
        iterReq.setIterator(iterator);
        iterReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);
        IterateResponse iterRes = (IterateResponse) client.send(iterReq);
```

EXAMPLE 2–18 CloseIterator Example *(Continued)*

```
        if (iterRes.getStatus().equals(StatusCode.SUCCESS)) {
            System.out.println("Found an iterator.");
        }
        for (PSO pso : iterRes.getPSOs()) {
            psos.add(pso);
        }
        iterator = iterRes.getIterator();

        // For this example, always close the iterator
        if (true) {
            CloseIteratorRequest closeIterReq = new CloseIteratorRequest();
            closeIterReq.setIterator(iterator);
            closeIterReq.setExecutionMode(ExecutionMode.SYNCHRONOUS);
            CloseIteratorResponse closeIterRes = (CloseIteratorResponse)
                client.send(closeIterReq);
            if (closeIterRes.getStatus().equals(StatusCode.SUCCESS)) {
                System.out.println("Closed iterator.");
                break;
            }
        }
    }
}
```

SPML Logging

Identity Manager can be configured to log SPML requests and responses. If Identity Manager receives a request or response that is known to the system, it writes pertinent information about the request to SPML log file. If it receives an unrecognized request, then it logs all the available information.

The SPML log is configured by editing the SPML configuration object. You cannot manage SPML logging from the administrator interface.

Access Log Configuration

Edit the following paramters in the SPML configuration object to enable and maintain logging.

- *AccessLogPath*. Specifies the full path to the active SPML access log file.
- *AccessLogMaxSize*. Specifies the maximum size, in kilobytes, of the active access log can grow to before it is archive. A value of 0 indicates there is no maximum file size.
- *AccessLogMaxArchiveFiles*. Specifies the maximum number of archive log files to preserve. A value of –1 indicates there is no limit. A value of 0 indicates to keep cycling the active log.

Access Log Example

The following sample shows a variety of logged requests.

```
# ListTargetsRequest
2009-02-03T01:16:12.083Z ListTargetsRequest requestID='rid[9964]' protocol='v2'
executionMode='synchronous' data='(accountId=Configurator,password=...)'
2009-02-03T01:16:12.089Z ListTargetsResponse requestID='rid[9964]' protocol='v2'
status='success' errorCode='null' targets='((spml2Person(objectclass(required=true),
accountId(required=true),credentials(required=true),firstname(required=null),
lastname(required=null),emailAddress(required=null))))' etime=40

# AddRequest
2009-02-03T01:16:12.582Z AddRequest requestID='null' protocol='v2'
executionMode='null' returnData='everything'
data='everything'2009-02-03T01:16:13.154Z
AddResponse requestID='Gen7951-1233623773152' protocol='v2' status='success'
errorCode='null' psoId='spml2Person:#ID#E413:20EBCB32F11:783B7992-:46D19511056F6FB3'
etime=578

# LookupRequest
2009-02-03T01:16:15.024Z LookupRequest requestID='null' protocol='v2'
 executionMode='synchronous' returnData='everything'
2009-02-03T01:16:15.130Z LookupResponse requestID='Gen7953-1233623775127' protocol='v2'
status='success' errorCode='null'
psoId='spml2Person:#ID#8092:20EBCB32F11:783B7992-:46D19511056F6FB3' etime=112

# ModifyRequest
2009-02-03T01:16:15.253Z ModifyRequest requestID='null' protocol='v2' executionMode='null'
returnData='everything' psoId='maurelius' modifications='(lastname=(Antoninus))'
2009-02-03T01:16:15.915Z ModifyResponse requestID='Gen7954-1233623775912' protocol='v2'
status='success' errorCode='null'
psoId='spml2Person:#ID#8092:20EBCB32F11:783B7992-:46D19511056F6FB3' etime=668
```

Configuring Identity Manager to Use SPML 2.0

This section describes how to configure Identity Manager to use SPML 2.0. The configuration involves the following:

Deciding Which Attributes to Manage

When configuring an Identity Manager server to use SPML 2.0, the first step is to decide which attributes you want to manage through your target.

Note – You can have more than one attribute in the target.

Decide which attribute sets, or object classes, the interface clients can employ to manage users in the Identity Manager instance using this interface. This set of attributes is a *PSO*. You must know how to map these attributes to and from a User view using a form.

This section describes how to configure a system using PSOs that contain the following attributes for a DSML object class called `spml2Person`:

- `accountId`
- `objectclass`
- `credentials`
- `firstname`
- `lastname`
- `emailAddress`

You must map these attributes to the User view.

This section also provides short examples that demonstrate how to manage PSOs using SPML 2.0 support in Identity Manager.

Identity Manager provides a sample set of SPML configuration objects in the `sample/spml2.xml` file. The `sample/spml2.xml` file is not imported when the repository is initialized, so you must manually import the file. See the contents of this file for detailed information.

Note – The `spml2ObjectClass` attribute is not present in the `User` schema by default. If this attribute is not already enabled, you must manually add the `spml2ObjectClass` attribute to your schema before Identity Manager can function as an SPML 2.0 server.

The `spml2ObjectClass` attribute has been defined in the `schema.xml` file supplied with Identity Manager, but the section where you add this attribute to the configuration is commented out. Assuming that your production schema is in a file derived from that original, you can uncomment that section, import or re-import the schema file, and restart Identity Manager to enable use of the SPML 2.0 feature.

After deciding on the format of a PSO, enable the service as described in the following sections. These sections also contain information about configuring the `web.xml` file and what features have been added for SPML 2.0.

Configuring the SPML2 Configuration Object

The sample/spml2.xml file contains an out-of-the-box configuration for SPML 2.0 support. You can import this file, or one derived from this file, to define the objects that Identity Manager needs to support SPML 2.0.

You can use the SPML2 configuration object type to change how SPML 2.0 support behaves or to extend the system.

Note – See "Extending the System" on page 62 for more information about extensions.

Configuring web.xml

If you are using a servlet container such as Tomcat, you can use the web.xml file to set up the openspmlRouter servlet, which handles SPML 2.0 requests.

Note – The web.xml file ships with a default installation, so no action is required.

The web.xml file contains an optional init-param parameter that you can use to open a monitor window (in Swing) that displays the flow of SPML 2.0 messages. You can use this window to monitor the flow of SPML 2.0 messages, which can be useful for debugging purposes.

The following example shows how to add the lisinit-param parameter.

EXAMPLE 2–19 Adding the init-param Parameter

```
<init-param>
    <param-name>monitor</param-name>
    <param-value>org.openspml.v2.util.SwingRPCRouterMonitor</param-value>
</init-param>
```

The next example contains a commented section and includes information about other init-param parameters.

EXAMPLE 2–20 Commented Example

```
<servlet>
    <servlet-name>openspmlRouter</servlet-name>
    <display-name>OpenSPML SOAP Router</display-name>
    <description>A router of RPC traffic - nominally SPML 2.0 over
        SOAP</description>
```

EXAMPLE 2–20 Commented Example *(Continued)*

```
    <servlet-class>
    org.openspml.v2.transport.RPCRouterServlet
    </servlet-class>
        <!-- The Router uses dispatchers to process SOAP messages. This is one that is in the toolkit that
        knows about SOAP. It has its own parameters, via naming convention. See below.
        -->
    <init-param>
        <param-name>dispatchers</param-name>
        <param-value>org.openspml.v2.transport.SPMLViaSoapDispatcher</param-value>
    </init-param>

        <!-- Turn on trace to have the servlet write informational messages to the log.
        -->
    <init-param>
        <param-name>trace</param-name>
        <param-value>false</param-value>
    </init-param>

       <!-- The SpmlViaSOAPDispatcher uses marshallers; there can be a chain, to move XML to SPML
       objects and back. -->
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberMarshaller</param-value>
    </init-param>

     <!-- Our marshaller (UberMarshaller) has its own trace setting; which doesn't really do anything in
     this release -->

    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers.UberMarshaller.trace</param-name>
        <param-value>true</param-value>
    </init-param>

        <!-- Finally, the dispatcher has a list of executors that actually implement the functionality. So, it
        sees a request, takes the SOAP envelope off, takes the body from XML to OpenSPML Request classes,
        and then asks the list of executors if they can process it. UberExecutor will redispatch the request to
        other executors. Those are specified in spml2.xml (Configuration:SPML2). -->

    <init-param>
        <param-name>SpmlViaSoap.spmlExecutors</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberExecutor</param-value>
    </init-param>
</servlet>
```

Configuring SPML Tracing

SPML provides options for turning on trace output so you can log Identity Manager's SPML traffic and diagnose problems.

For more information about tracing SPML, see Chapter 5, "Tracing and Troubleshooting," in *Sun Identity Manager 8.1 System Administrator's Guide*.

Extending the System

You extend the schema by modifying the configuration object, and you can add executors for requests by changing the section. Using forms, you can map DSML to Views and back.

It is less obvious, but you can also replace the dispatcher, marshaller, and the UberExecutor, with those of your own devising.

- If you do not want to use SOAP, replace the dispatcher in the first case.
- If you do not want to use HTTP, replace the router with a different kind of servlet.
- If you want different XML parsing, replace the marshaller with your own.

SPML 2.0 provides a wide-open array of pluggability, which is due to Identity Manager's use of the OpenSPML 2.0 Toolkit. The following figure shows the OpenSPML 2.0 Toolkit architecture.

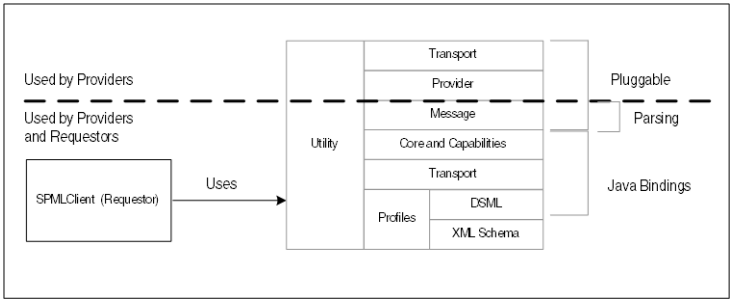

FIGURE 2–1 OpenSPML 2.0 Toolkit Architecture

SPML Connector

Identity Manager provides a connector to manage communications to the resource. See the *Sun Identity Manager 8.1 Resources Reference* for information about implementing this feature.

Index

D

E

F

G

I

S

T

U

V

W